Edition de documents multi-langages sous Mentor-Rapport

B. Melese

INRIA

CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(3)954.90.20

Rapports Techniques

N° 54

ÉDITION DE DOCUMENTS MULTI-LANGAGES SOUS MENTOR-RAPPORT

Bertrand MÉLÈSE

Mai 1985

Edition de Documents Multi-langages sous Mentor-Rapport

Résumé

La plupart des documents techniques associent plusieurs langages, ce qui complique considérablement leur traitement. le système *Mentor-Rapport* est une extension l'environnement de programmation *Mentor* qui permet la manipulation de documents multi-langages et ouvre par là de nouvelles perspectives dans le traitement des textes complexes.

Le système Mentor-Rapport s'applique de façon particulièrement intéressante au domaine du génie logiciel. En effet, un logiciel est composé de plusieurs parties: spécifications, documentation, exemples, jeux de tests, programmes; chacune de ces parties est écrite dans un langage qui lui est propre et peut elle-même associer plusieurs langages. Actuellement, dans la plupart des cas, les différentes parties d'un logiciel sont créées, modifiées et maintenues indépendamment les unes des autres ce qui complique la tâche de l'équipe de développement et conduit souvent à des états incohérents. Nous montrons qu'un logiciel peut etre considéré comme document multi-langages ce qui, en présence d'outils de traitement adéquats, permet de maintenir plus facilement, et de façon automatisable, la cohérence entre ses différentes parties.

Editing Multi-language Documents with Mentor-Rapport

Abstract

The manipulation of technical documents is made harder when several languages are mixed in these documents. The Mentor-Rapport system is based on the Mentor programming environment and allows the manipulation of multi-language documents. This brings new possibilities for the manipulation of complex documents.

Mentor-Rapport is well adapted to support software engineering activities: a software system is made of several components (specifications, documentation, examples, tests, programs, ...), each of these being written in its own language and possibly being made of several languages. In most of the cases today, components of a software are created, modified and maintained independantly. This makes these tasks difficult and often leads to incoherent states. We show that the ability to represent a software system by a multi-language document and to manipulate it with adequate tools makes these tasks easier and makes it possible to automatically maintain the coherence of the system.

Edition de Documents Multi-langages sous Mentor-Rapport

Bertrand Mélèse
I.N.R.I.A.
Domaine de Voluceau - Rocquencourt
B.P. 105 - 78153 Le Chesnay Cedex

Introduction

La plupart des documents techniques associent plusieurs langages, ce qui complique considérablement leur traitement. Le système *Mentor-Rapport* est une extension de l'environnement de programmation *Mentor* qui permet la manipulation de documents multi-langages et ouvre par là de nouvelles perspectives dans le traitement des textes complexes.

Considérons par exemple un manuel du langage de programmation Pascal: on y trouve des paragraphes en langage naturel, des définitions syntaxiques en BNF et/ou sous forme de chartes syntaxiques, des exemples en Pascal, des figures expliquant le déroulement des instructions; on y trouve également les descriptions sémantiques des éléments du langage, souvent exprimées en langage naturel mais qui peuvent être écrites dans un formalisme adapté. Tous ces composants du manuel sont organisés en chapitres, sections, sous-sections, annexes, bibliographie. Un programme Pascal lui-même, s'il est correctement documenté, peut être vu sous deux angles: un programme compilable avec des commentaires ou un document publiable dont certains paragraphes sont écrits en Pascal.

Le système Mentor-Rapport s'applique de façon particulièrement intéressante au domaine du génie logiciel. En effet, un logiciel complet est, ou devrait être, composé de plusieurs parties: spécifications, documentation, exemples, jeux de tests, programmes; chacune de ces parties est écrite dans un langage qui lui est propre et peut elle-même associer plusieurs langages. Actuellement, dans la plupart des cas, les différentes parties d'un logiciel sont créées, modifiées et maintenues indépendamment les unes des autres ce qui complique la tâche de l'équipe de développement et conduit souvent à des états incohérents. Nous verrons qu'un logiciel devrait être considéré comme un seul (gros) document multi-langages ce qui, en présence d'outils de traitement adéquats, permet de maintenir plus facilement, et de façon automatisable, la cohérence entre ses différentes parties.

On peut classer les langages mis en jeu en deux catégories: le langage naturel et les langages structurés. A l'intérieur même d'un langage structuré (langage de programmation, de spécification, de description de figures, de circuits ou de formules, langage de description de la structure externe d'un rapport technique etc.) on trouve deux types de composantes: celles qui ont une structure syntaxique forte reflétant de près la sémantique, par exemple, les instructions d'un langage de programmation, et les composantes dont la structure syntaxique est beaucoup plus libre vis-à-vis de la sémantique, tels les commentaires d'un programme ou les paragraphes en langage naturel dans un rapport technique.

Les systèmes dirigés par la syntaxe sont bien adaptés au traitement des composantes à structure syntaxique forte. En revanche, les éditeurs de texte sont plus performants pour l'édition du langage naturel et des composantes à structure syntaxique faible des langages structurés.

Pour être un outil général et commode à utiliser, un environnement de programmation doit donc permettre de manipuler des documents multi-langages en choisissant, pour chaque composante du document, le mode d'édition le mieux adapté. Il doit également permettre *d'effectuer des calculs de cohérence* entre les différentes composantes de ces documents.

Le but de cet article est de montrer qu'avec Mentor-Rapport:

i) L'édition de documents multi-langages est naturelle grâce à une interface utilisateur qui choisit automatiquement l'éditeur le mieux adapté à la composante éditée: éditeur syntaxique pour les composantes à structure syntaxique forte, éditeur de texte pour les autres.

ii) L'interface utilisateur de l'édition syntaxique est améliorée grâce à la possibilité d'utiliser un éditeur de texte, en complément de l'éditeur syntaxique, pour l'édition de composantes structurées lorsque l'utilisateur juge cela plus commode (voir section 1.2).

iii) L'utilisateur peut obtenir plusieurs vues des documents multi-langages; en particulier, il peut les faire imprimer sur un périphérique à haute résolution: photocomposeuse, imprimante à laser etc. L'idée d'avoir plusieurs vues d'un même document, par exemple de voir un programme bien documenté comme un document publiable dont certains paragraphes sont des fragments de programmes, est à la base du système WEB [Knuth 82] et est généralisée dans le système Pecan [Reiss 84] à d'autres types de vues (organigramme, flot des données).

Nous verrons que, dans ce domaine, Mentor-Rapport est plus performant que WEB et comparable à Pecan même s'il n'autorise pas, dans sa version actuelle sur terminal alphanumérique, de représentations graphiques. En effet, dans WEB, toutes les parties des documents sont éditées sous forme textuelle et l'utilisateur doit indiquer, par des commandes insérées dans le texte, les séparations entre les parties et les ordres de mise en page. A l'inverse, dans Mentor-Rapport, la représentation structurée des documents, la manipulation des différents composants avec l'éditeur le mieux adapté et l'insertion automatique des ordres de mise en page (pour piloter un système de composition de texte) affranchissent l'utilisateur de tâches fastidieuses.

iv) Les acquis de deux domaines sont exploités. Le premier est l'édition syntaxique de programmes, qui, dans le cas de Mentor-Rapport, est généralisée à l'édition de documents multi-langages. Le deuxième est l'aide à la préparation de documents dont le but est de fournir une interface de haut niveau entre les systèmes de composition de texte et leurs utilisateurs. Ce domaine est illustré par de nombreux systèmes [Sigplan 81].

Ce dernier point mérite d'être développé: de nombreux systèmes d'aide à la préparation de documents mettent en avant la nécessité d'avoir une représentation structurée des documents [Reid 80, 83, Wood 81, Stromfors 81, Walker 81, Hammer 81, Allen 81], ainsi que des formules mathématiques et des figures qu'ils contiennent [Kernighan 81, Quint 83, Van Wyk 81] pour pouvoir fournir une aide efficace aux utilisateurs. Si certains de ces auteurs évoquent brièvement le rapport entre leur système et l'édition syntaxique [Quint 83, Walker 81, Wood 81], aucun ne franchit le pas et n'utilise un éditeur syntaxique comme point de départ. L'existence d'une frontière nette entre ces deux domaines provient du fait que peu d'éditeurs syntaxiques sont suffisamment généraux pour permettre l'édition de documents complexes incluant du texte, des programmes, des formules, des figures etc. Mentor-Rapport franchit ce pas.

La section 1 est un bref exposé des principales caractéristiques du système *Mentor*. Dans cette section, nous exposons en particulier les raisons pour lesquelles la possibilité d'édition textuelle à l'intérieur d'un éditeur syntaxique nous paraît importante pour le confort de l'utilisateur (section 1.2) et nous expliquons comment la communication entre Mentor et un éditeur de texte est réalisée (section 1.3). Dans la section 2, nous décrivons l'environnement *Mentor-Rapport*.

1. Principales caractéristiques du système Mentor

1.1. Concepts de base

Le but de cette section est de donner une description brève, mais suffisante pour lire la suite de l'article, des concepts de base du système Mentor. Une définition complète de ces concepts peut être trouvée dans [Donzeau-Gouge 83, 84a, 84b].

Le système Mentor [Donzeau-Gouge 83, 84a, 84b] est un environnement de programmation basé sur le concept d'édition structurée guidée par la syntaxe du langage auquel les objets manipulés appartiennent. Il existe d'autres systèmes basés sur ce concept [Sigplan 84]. Parmi les plus connus on peut citer: Gandalf [Feiler 81, Gandalf 84], le "Cornell Program Synthesizer" [Teitelbaum 81, Reps 82], Adèle [Mossière 82, Estublier 83], Pecan [Reiss 84]. Ces systèmes partagent les idées suivantes avec Mentor:

- Représentation arborescente, dite *représentation abstraite,* des objets manipulés. La forme textuelle des objets, utilisée pour être affichée sur l'écran ou sauvegardée dans un fichier, est calculée à partir de l'arbre par un processeur nommé *décompilateur* ou *paragrapheur.*
- Utilisation d'une *syntaxe abstraite* [Donzeau-Gouge 83, 84b] qui définit les arbres légaux dans le langage édité et qui contrôle que ces arbres restent légaux au cours des transformations appliquées pendant une session d'édition.
- Existence d'un ensemble de commandes pour la manipulation interactive des représentations abstraites.

Les principaux points par lesquels Mentor se distingue des autres systèmes sont les suivants:

- Mentor est *programmable.* Le langage de commande de Mentor est un langage de programmation, appelé *Mentol,* spécialement conçu pour l'écriture de procédures de manipulation et de transformation de la représentation abstraite des objets édités. Du point de vue de l'utilisateur, les procédures *Mentol* sont invoquées interactivement, de la même façon que les commandes de base du système. Le langage *Mentol* est utilisé pour enrichir l'environnement Mentor par des commandes spécifiques à une application particulière [Mélèse 80, 81].
- Mentor est *extensible* à de nouveaux langages. Un langage de spécification, appelé *Métal,* permet d'introduire dans Mentor les langages que l'on souhaite éditer [Mélèse 82, Kahn 83]. De nombreux langages ont déjà été introduits dans le système: Pascal, Ada[1], LTR3, LTR-V2, Lisp, Chill etc. L'introduction d'un nouveau langage est à la portée d'un utilisateur non spécialiste du système. En Métal, seules les caractéristiques *syntaxiques* des langages à introduire dans Mentor sont définies. Un second langage de spécification, appelé *Typol* permet de spécifier les caractéristiques *sémantiques* des langages introduits (vérificateurs de types, interpréteurs) [Despeyroux 84].

---

[1] Ada is a trademark of the US DoD Ada Joint Program

- Mentor est *multi-langages.* Il est possible de manipuler, au cours d'une même session Mentor, des objets appartenant à des langages différents. Par les mécanismes *d'annotations* et de *portes* [Donzeau-Gouge 83, 84b] il est également possible de manipuler des objets dans lesquels plusieurs langages interviennent, c'est-à-dire des *documents multi-langages* (voir l'introduction de cet article).

  Les documents multi-langages sont représentés par des arbres dans lesquels plusieurs syntaxes abstraites coexistent. Lorsque l'utilisateur se déplace d'une partie de son document à une autre, qui est dans un langage différent, Mentor change automatiquement d'environnement pour s'adapter à ce nouveau langage. L'utilisateur a donc toujours en face de lui un éditeur syntaxique adapté au langage de la composante éditée.

  Nous reviendrons sur ce point dans la section 2. En effet, Mentor-Rapport utilise l'aspect multi-langage de Mentor en le complétant par l'appel d'un éditeur de texte, lorsque la composante à éditer est en langage naturel, et par une interface utilisateur appropriée.

- Mentor est un système *ouvert* et *extensible.* Toutes ses fonctionnalités sont accessibles à des programmes extérieurs au système grâce à une interface constituée de procédures et de fonctions écrites en Pascal. Un utilisateur peut donc ajouter de nouveaux processeurs au système ou même utiliser Mentor comme un environnement de base pour l'implémentation d'un système spécialisé [Migot 83, Schroeder 84]. Cette interface étant réalisée en Pascal, les programmes qui l'utilisent doivent être écrits dans un langage à partir duquel il est possible d'appeler des sous programmes Pascal. A titre d'exemple, indiquons que le langage de spécifications sémantiques *Typol* utilise une liaison entre Mentor et PROLOG.

1.2. Le rôle de l'édition de texte dans Mentor

Nous indiquons dans cette section les raisons pour lesquelles l'interface utilisateur d'un éditeur syntaxique se trouve améliorée par la possibilité d'utiliser localement un éditeur de texte pour certaines opérations. En effet, même lorsque le langage édité est structuré, certaines transformations sont à caractère textuel (voir 1.2.1 et 1.2.2 ci-dessous) et d'autres sont rendues plus intuitives une fois ramenées en mode textuel (voir 1.2.3).

1.2.1. Edition des composantes non structurées d'un langage

Tous les langages de programmation contiennent des composantes non structurées (commentaires, chaînes de caractères, identificateurs) qu'il est naturel d'introduire et de modifier avec un éditeur de texte. Par exemple, le remplacement d'un caractère par un autre dans une ligne de commentaire, un identificateur ou une chaîne de caractères est une opération purement textuelle. Pour effectuer une telle modification en mode syntaxique, l'utilisateur doit saisir la nouvelle ligne de commentaire, le nouvel identificateur ou la nouvelle chaîne in-extenso puisque ce sont des atomes indivisibles de la structure. S'il en a la possibilité, l'utilisateur juge souvent plus commode d'appeler un éditeur de texte pour effectuer la modification.

1.2.2. Saisie de nouveau code

La saisie de nouveau code est un autre exemple pour lequel l'édition textuelle est en général plus adaptée que l'édition syntaxique. En effet, en mode syntaxique, la saisie s'effectue par l'intermédiaire d'un système de menus: l'utilisateur choisit dans le menu proposé la structure qu'il désire insérer à un endroit donné de son programme. Le système propose ensuite un nouveau menu pour chacun des composants de cette structure. Dans ce mode, seules les unités lexicales du langage (identificateurs, chaîne de caractères, entiers, réels, etc), sont saisies directement par l'utilisateur [Migot 85]. Ce mode de saisie est utilisé systématiquement dans des systèmes tels que Gandalf et le "Cornell Program Synthesizer" et facilite la tâche des utilisateurs débutants ou peu familiers avec le langage de programmation utilisé. L'expérience montre que ce mode de saisie (appelé *mode débutant* sous Mentor) est rapidement perçu comme étant très lourd par les utilisateurs expérimentés.

Signalons que, dans Mentor, le code peut être saisi directement à la console sans aucun intermédiaire. Le principal défaut de ce mode est qu'il ne permet pas de récupérer le texte entré pour le modifier en cas d'erreurs. Par contre, sa simplicité en fait le mode favori des utilisateurs confirmés pour la saisie de courts fragments de code.

1.2.3. Edition de structures profondes

Dans les langages de programmation, les expressions arithmétiques ou logiques sont des cas typiques de structures pour lesquelles une représentation textuelle courte peut engendrer une représentation arborescente profonde. En Mentor par exemple, lorsque le langage manipulé est Pascal, l'expression A+(D mod 13 -B * (C-1) div 2) est représentée par l'arbre de la figure 1. En général, les utilisateurs préfèrent éditer de telles expressions en mode texte plutôt que directement sur l'arbre qui les représente.

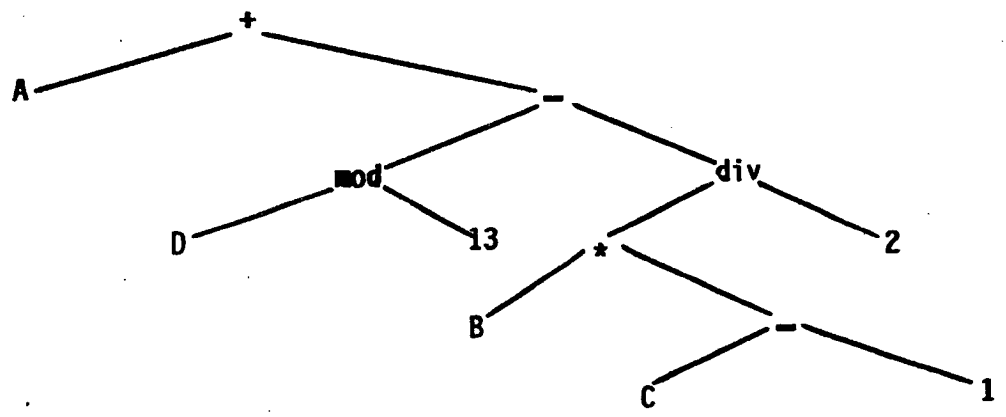

figure 1: Représentation arborescente de l'expression A+(D mod 13 -B * (C-1) div 2) sous Mentor

Insistons sur le fait que, dans Mentor, la représentation interne des expressions est arborescente, et non textuelle comme dans certains éditeurs syntaxiques (le "Cornell Program Synthesizer" en particulier), ce qui assure l'uniformité de la structure interne indispensable dès qu'on envisage de faire des calculs symboliques sur cette structure.

1.3. Réalisation de la communication entre Mentor et un éditeur de texte

La communication entre Mentor et un éditeur de texte est réalisée grâce au décompilateur (ou paragrapheur) et à l'analyseur-constructeur d'arbres (figure 2). Rappelons que le *décompilateur* est le processeur qui crée le texte associé à la représentation arborescente d'un objet, tandis que *l'analyseur-constructeur d'arbres* est le processeur qui effectue l'analyse syntaxique d'un texte et crée la représentation arborescente correspondante.

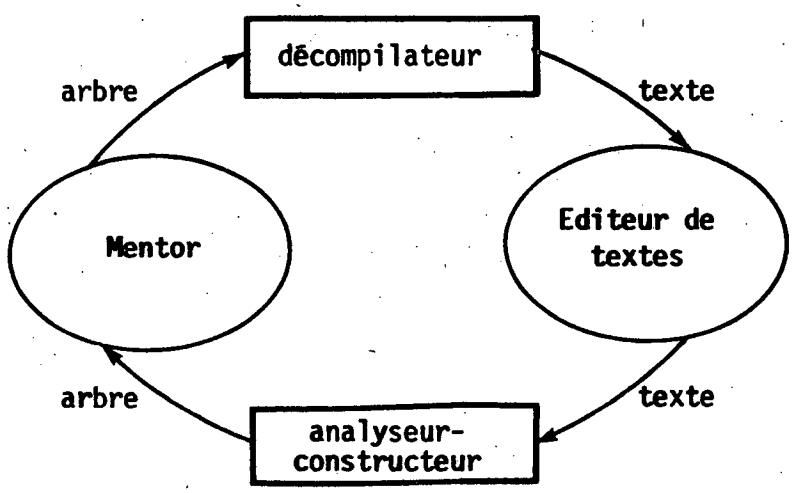

figure 2: Communication entre Mentor et un éditeur de texte

Grâce à ces deux processeurs, le principe de la communication est très simple: pour transmettre un objet de Mentor vers l'éditeur de texte il suffit de créer, par décompilation, le texte correspondant et de l'envoyer vers l'éditeur de texte. De la même façon, pour transmettre un objet de l'éditeur de texte vers Mentor on envoie le texte de cet objet dans l'analyseur-constructeur qui crée l'arbre correspondant; cet arbre est ensuite accroché à l'arbre principal par les primitives standard de Mentor.

Ce mode de communication a les trois avantages suivants:

i) il permet de transmettre n'importe quel fragment de programme grâce au fait que l'analyseur-constructeur est *multi-points d'entrées,* c'est-à-dire capable d'analyser n'importe quel sous ensemble du langage considéré (un identificateur, une expression, une instruction, etc).

ii) Il est adaptable facilement à tous les éditeurs de texte; nous pensons en effet important que l'utilisateur puisse choisir l'éditeur de texte qui lui est familier plutôt que de lui en imposer un autre, spécifique au système.

iii) il est simple à mettre en oeuvre: l'utilisateur appelle une commande qui le connecte à l'éditeur de texte choisi.

1.4. Remarques

A) Sur l'analyse syntaxique

Le mode de communication utilisé repose sur le fait que Mentor possède un analyseur syntaxique pour chaque langage. Actuellement, ces analyseurs sont incapables de prendre en compte des fragments de texte dans lesquels plusieurs langages coexistent. L'utilisation de l'éditeur de texte est donc limitée aux fragments de textes qui ne contiennent qu'un seul langage.

B) Sur l'intégration des outils

Nous nous sommes appliqués à rendre la communication entre Mentor et les éditeurs de texte indépendante de l'éditeur de texte utilisé, pour ne pas imposer aux utilisateurs un éditeur particulier. Cependant, cette approche pose le problème de l'intégration étroite de ces outils; en effet, le principal désagrément, lors de l'utilisation de cette communication, provient du fait que les deux systèmes sont totalement disjoints; cela se traduit parfois par un mode de travail discontinu car il est difficile de modifier, dans un même appel de l'éditeur, les textes qui correspondent à plusieurs sous-arbres disjoints. Dans le cadre de Mentor-Rapport, cette situation est améliorée par l'intégration de commandes globales de substitution de chaînes de caractères (voir section 2.2.3).

Au cours du développement du système Mentor, nous avons dû choisir, à plusieurs occasions, entre la création de nouveaux outils et la réutilisation d'outils existants; ce fut le cas pour les compilateurs, l'analyseur syntaxique général, les éditeurs de texte et, plus récemment, les systèmes de composition de texte utilisés par Mentor-Rapport. La première solution, la création de nouveaux outils, a l'avantage de permettre une bonne intégration entre ces outils et de simplifier les transports du système d'un type d'ordinateur à un autre. Ses inconvénients majeurs sont, d'une part, le temps nécessaire au développement de ces outils et, d'autre part, le fait que les outils que nous aurions développés n'auraient pas été aussi performants que ceux développés par des spécialistes de ces différents domaines (compilation, analyse syntaxique, édition de texte). Nous avons donc toujours choisi la deuxième solution, la réutilisation d'outils existants, en nous attachant à rendre l'interface entre Mentor et ces outils aussi simple que possible, pour pouvoir facilement l'adapter à d'autres outils équivalents.

Dans le cas de l'interface entre Mentor et les éditeurs de texte le problème posé aujourd'hui est donc l'amélioration de la transparence de la communication. Actuellement, nous réalisons une version de Mentor adaptée au fonctionnement sur terminal à haute résolution (écran "bit map"). Dans cette version, il sera possible d'avoir des fenêtres contenant en permanence une session de l'éditeur de texte. Le simple déplacement de la souris d'une fenêtre de Mentor à une fenêtre de l'éditeur de texte aura alors pour effet de changer le mode d'édition.

2. Mentor-Rapport

Rapport est un langage de description de documents sous forme structurée, en termes de chapitres, sections, sous-sections, titres, paragraphes, bibliographie etc. Aucune tentative n'est faite pour modéliser la structure du langage naturel: toutes les composantes textuelles (titres, paragraphes en langage naturel, etc) sont des atomes du langage Rapport qui sont systématiquement saisis et modifiés à l'aide d'un éditeur de texte.

Mentor-Rapport est l'environnement de manipulation des documents décrits en Rapport. Mentor-Rapport est construit à partir de Mentor et hérite donc de toutes les fonctionnalités de ce système. Dans cette section nous décrivons les fonctionnalités importantes propres à Mentor-Rapport en insistant sur l'édition de documents multi-langages et la mise en page des documents. Quelques commandes représentatives de l'environnement sont décrites dans la section 2.2. Pour une description plus exhaustive et plus détaillée des structures et des commandes disponibles en Mentor-Rapport, le lecteur se reportera au manuel d'utilisation [Mélèse 83] et au chapitre 11 de la documentation du système Mentor [Mélèse 85].

2.1. Documents Multi-langages sous Mentor-Rapport: Notion de "Portes".

Nous décrivons dans cette section le mécanisme des *portes*, déjà évoqué dans la section 1.1, qui est un des mécanismes de Mentor pour représenter les documents multi-formalismes et qui est utilisé par Mentor-Rapport.

Les *portes* sont des atomes d'un langage dont la particularité est d'avoir une structure interne dans un autre langage (figure 3). Un tel atome est une frontière entre deux langages d'où le nom de *porte*. Bien entendu, le langage interne d'une porte peut lui-même avoir des portes vers d'autres langages. Il n'y a aucune limitation sur le nombre de niveaux d'emboitements des langages par l'intermédiaire des portes.

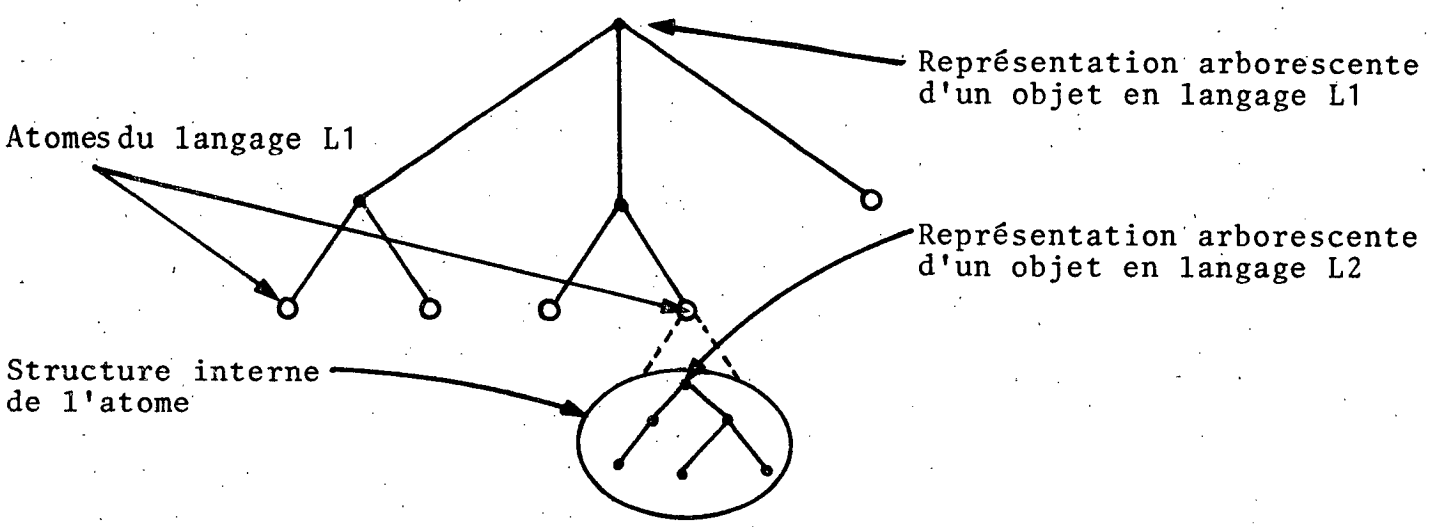

figure 3: Les portes

Lorqu'une porte est franchie, l'environnement d'édition est automatiquement adapté au nouveau langage, langage interne ou externe selon le sens dans lequel la porte a été franchie. Au cours d'une session Mentor-Rapport, le nom du langage courant est toujours indiqué en haut à droite de l'écran pour informer l'utilisateur des changements d'environnement.

Les applications des portes sont nombreuses. Certaines sont décrites en détails dans [Donzeau-Gouge 84b]. Nous donnons ci-dessous un exemple d'utilisation des portes dans le contexte de Mentor-Rapport.

Considérons à nouveau un manuel du langage de programmation Pascal, et plaçons nous dans le cas d'un manuel idéal, de notre point de vue, c'est-à-dire contenant une description formelle de la sémantique de Pascal, ainsi qu'une syntaxe abstraite (pour standardiser l'édition syntaxique des programmes Pascal). Sous Mentor-Rapport, un tel manuel pourrait être représenté de la façon suivante:

- la structure générale du manuel est définie en Rapport: en-tête, titres, chapitres, sections, sous-sections, paragraphes, annexes, bibliographie etc;

Tous les composants suivants sont des paragraphes, et donc des atomes, du point de vue du langage Rapport, et sont des portes vers d'autres langages.

- Les paragraphes en langage naturel sont des portes vers un langage très simple, le langage du texte libre structuré en lignes;
- Les définitions syntaxiques sont des portes vers le langage de la BNF (Backus-Naur-Form) ou vers le langage Métal [Mélèse 82, Kahn 83];
- Les figures et les diagrammes de syntaxe sont des portes vers un langage de description de figures, par exemple Flip [Kahn 81a], PIC [Kernighan 81], ou Ideal [Van Wyk 81];
- Les définitions de la syntaxe abstraite sont des portes vers le langage Métal;
- Les exemples en Pascal sont des portes vers le langage Pascal;
- Les descriptions sémantiques sont des portes vers un langage approprié, par exemple Typol [Despeyroux 84] ou ML [Milner 84].

Lorsque l'utilisateur décide de créer ou de modifier un paragraphe en langage naturel, Mentor-Rapport appelle automatiquement l'éditeur de texte en utilisant le mécanisme de communication décrit dans la section 1.3. Notons que, dans le cas du langage naturel, l'analyseur syntaxique utilisé est simpliste: il découpe le texte en lignes, la structure arborescente des paragraphes textuels étant simplement une liste de lignes.

2.2. Exemples de commandes disponibles dans l'environnement Mentor-Rapport

Un des buts poursuivis dans la conception de Mentor-Rapport est de rendre cet environnement utilisable par des gens qui ne sont pas spécialistes du système Mentor. Pour cela nous nous sommes attachés à ne fournir qu'un petit nombre de commandes générales qui proposent des menus à l'utilisateur. Bien entendu, toutes les commandes standard de Mentor sont disponibles en Mentor-Rapport pour les utilisateurs confirmés.

Nous décrivons ci-dessous quelques commandes représentatives de l'environnement Mentor-Rapport.

2.2.1. Commandes de création et modification: *créer* et *édit*

La commande *créer* est utilisée pour la création de toutes les structures disponibles en Rapport: chapitres, sections, sous-sections, paragraphes (qui sont toujours des portes), bibliographie etc. L'utilisateur choisit dans un menu la structure qu'il souhaite créer. Lorqu'il choisit de créer un paragraphe le système demande quel est le langage interne de ce paragraphe.

Lorque l'utilisateur demande la création d'un paragraphe textuel il est connecté automatiquement à l'éditeur de texte. Le texte saisit sera ensuite inséré automatiquement dans le document. Lorque l'utilisateur demande la création d'un paragraphe dans un langage structuré le système propose trois modes de saisie: la lecture d'un fichier existant, le mode débutant (voir section 1.2.2) et le mode expert. Ce dernier mode consiste simplement à laisser l'utilisateur, supposé expert, se débrouiller avec les commandes de saisie standard de Mentor.

La commande *édit* est utilisée pour modifier une partie quelconque du document. La demande de modification d'une partie textuelle connecte l'éditeur de texte à cette partie. La demande de modification d'un paragraphe en langage structuré propose un choix entre le mode débutant et le mode expert et se charge de reconnaître le langage interne de ce paragraphe. Les modifications de la structure globale du document sont assistées par d'autres commandes que nous ne décrirons pas ici: déplacement d'un chapitre, d'une section ou d'une sous-section par exemple.

2.2.2. Commandes de déplacement dans la structure du document

La commande *chercher* permet de se positionner directement sur un chapitre. Les commandes *psuiv, ppre, pder* et *bib* permettent respectivement de se positionner sur le paragraphe qui suit la position courante, celui qui précède cette position, le dernier paragraphe ou sur la bibliographie associée au document. Les commandes *corps* et *titre* sont utilisées respectivement pour se positionner sur le corps et sur le titre du chapitre ou de la section courante.

2.2.3. Commandes globales de substitution dans les composantes textuelles

Le fait que Mentor-Rapport et l'éditeur de texte soient totalement disjoints rend difficiles les modifications globales dans les composantes textuelles: il faudrait, en principe, éditer séparement chacune de ces composantes. Une telle contrainte est inacceptable et nous avons donc intégré à Mentor-Rapport des commandes de substitution de chaînes de caractères compatibles avec la structure et l'aspect multi-langages des documents: ces commandes n'effectuent les substitutions que dans les composantes textuelles et leur portée est déterminée par la structure arborescente du document.

Les commandes de substitution *sg, sl,* et *sq* demandent à l'utilisateur la chaîne à remplacer (C1) et la chaîne remplaçante (C2). La commande *sg* remplace par C2 toutes

les occurences de C1 qui se trouvent entre la position courante et la fin du document. La commande *sl* ne remplace que les occurences de C1 qui se trouvent dans la partie du document determinée par le sous arbre dont la racine est la position courante. Les commandes *sg* et *sl* sont donc équivalentes quand la position courante est la racine de l'arbre qui représente le document. La commande *sq* a le même comportement que *sl* mais demande, sur chaque occurence de C1, si le remplacement doit être effectué ou pas.

2.2.4. Division des documents en plusieurs parties

Lorque la taille des documents édités devient grande, il est nécessaire de les diviser en plusieurs parties, éditables séparément, tout en conservant la cohérence de la structure interne. La commande *séparer* prend une partie du document et en fait une unité éditable indépendamment. Les seules unités séparables sont les chapitres, les sections et les sous-sections. Les unités séparées peuvent elles-mêmes avoir des sections ou des sous-sections séparées.

Lorqu'un document est divisé, Mentor-Rapport conserve suffisamment d'information dans chaque partie pour permettre de simuler, aux yeux de l'utilisateur, l'édition du document complet. Il est aussi possible, avec la commande *regrouper,* de remettre une unité séparée à sa place originale.

2.2.5. Représentations abrégées des documents

Lorsqu'on édite un document quelconque (programme, rapport technique etc) il est souvent utile de pouvoir visualiser sa structure globale, ou celle de l'une de ses parties, par exemple pour trouver rapidement l'endroit que l'on doit modifier. Or, la taille des écrans ne permet d'afficher qu'une petite portion du document. Mentor-Rapport possède trois mécanismes, complémentaires les uns des autres, pour afficher des représentations abrégées qui mettent en évidence la structure globale des documents.

i) Le premier mécanisme est *l'affichage partiel des arbres,* disponible de façon standard dans Mentor. Ce mécanisme est basé sur la profondeur d'emboîtement des sous arbres par rapport à la position courante: tout sous arbre dont la profondeur d'emboîtement est supérieure à un paramètre, appelé *niveau de détails,* est remplacé, à l'affichage, par un symbole d'abréviation. Le niveau de détails peut être modifié dynamiquement par l'utilisateur.

L'affichage partiel des arbres est satisfaisant lorsqu'on manipule un langage de programmation mais se révèle insuffisant dans le cadre de Mentor-Rapport. En effet, les paragraphes textuels sont représentés par des listes de lignes et sont donc, avec ce mécanisme, soit affichés en entier, soit remplacés globalement par un symbole d'abréviation.

ii) Le second mécanisme, mis en oeuvre par les commandes *pc* (paragraphes en forme courte) et *pl* (paragraphes en forme longue), complète le mécanisme d'affichage partiel. Après l'exécution de la commande *pc* les paragraphes textuels de plus de quatre lignes sont affichés sous une forme abrégée, composée de leurs deux premières lignes suivies de points de suspension, ce qui est en général suffisant pour caractériser

un paragraphe particulier. L'exécution de la commande *pl* rétablit l'affichage complet des paragraphes textuels.

iii) La table des matières est une représentation abrégée utile interactivement, car elle donne une vue globale de la structure du document, et nécessaire pour les documents destinés à être imprimés. La commande *plan* affiche la table des matières d'un document. La numérotation des chapitres, sections et sous-sections est calculée à chaque appel de cette commande. Elle est donc toujours correcte quelles que soient les modifications apportées à la structure générale du document (échange, fusion, destruction de chapitres ou de sections etc).

2.2.6. Points de vue multiples sur un document

Lorque le document édité est complexe, un logiciel complet par exemple, il est indispensable de pouvoir le regarder selon différents points de vue en fonction des traitements qu'on souhaite lui appliquer.

Prenons l'exemple d'un compilateur représenté par un document multi-langages dans lequel il y a quatre types de composants: spécifications, documentation, programmes et jeux de tests. Supposons que ce document ait une structure hiérarchique, comme cela est possible en Mentor-Rapport, et que chaque chapitre, section et sous-section, traite d'un aspect particulier du compilateur. On aurait par exemple quatre chapitres:

Chapitre 1: Analyse Lexicale
Chapitre 2: Analyse Syntaxique
Chapitre 3: Vérification des Types
Chapitre 4: Production de Code

Chaque chapitre est ensuite divisé en plusieurs sections et sous-sections:

Chapitre 2: Analyse Syntaxique
2.1. Analyse Syntaxique des Déclarations
2.1.1. Déclarations de constantes
2.1.2. Déclarations de types
2.1.3. Déclarations de variables
...
2.2. Analyse Syntaxique des Expressions
...
2.3. Analyse Syntaxique des Instructions
2.3.1 Les instructions simples
...
2.3.2 Les instructions composées
...

Dans chaque section ou sous-section on trouve la documentation, les spécifications, les jeux de tests et les programmes qui correspondent à la partie du compilateur qui y est traitée.

Une fois construit, ce document doit servir à plusieurs fins qui correspondent à différents points de vue sur le document. Pour produire un manuel du langage on ne voudra garder que les parties *documentation* et *spécifications* et en faire un document publiable. Pour produire le compilateur lui même on ne gardera que les *programmes* qu'on compilera. Pour tester le compilateur ainsi produit on ne gardera que les *jeux de tests*. Il est facile de produire de telles représentations partielles, de tels points de vue, d'un même document: chaque type de composant peut être rendu visible ou invisible, soit pour l'affichage, soit pour la sortie dans des fichiers, soit encore pour des processeurs particuliers.

2.2.7. Modèles de documents

Le système Mentor offre la possibilité de créer et de manipuler des documents incomplets, c'est-à-dire des documents dont certaines parties n'ont pas encore été définies. Un document incomplet est représenté par un arbre dont certains sous arbres sont remplacés par des atomes spéciaux appelés *métavariables*. Lorsque le langage manipulé est un langage de programmation, cette possibilité est utilisée pour définir des schémas de programmes, qui, par la suite, peuvent être complétés de différentes façons pour obtenir des programmes adaptés à une situation particulière [Donzeau-Gouge 84a].

Sous Mentor-Rapport cette technique est utilisée pour définir des modèles de documents dont le format est toujours le même: lettres, contrats, thèses, formulaires, rapports techniques etc. Ces modèles peuvent ensuite être regroupés dans des bibliothèques et être utilisés comme point de départ pour la rédaction de documents particuliers. Par exemple, la commande *df* sera utilisée pour la création d'un document en français sur le modèle *rapport technique*. Elle construit le squelette du rapport en posant des questions à l'utilisateur: nom du rapport, titre, nom des auteurs, adresses des auteurs etc.

Il est aussi envisageable de construire la représentation interne d'un document, selon un certain modèle, par analyse syntaxique de la représentation textuelle de ce document. L'avantage d'une telle possibilité serait de pouvoir introduire dans Mentor-Rapport des documents qui n'ont pas été construits dès le départ avec ce système. Des travaux intéressants de ce point de vue sont présentés dans [Verges-Escuin 73].

2.3. Mise en page des documents: la commande *compose*

Une fonctionnalité importante de Mentor-Rapport est la possibilité qu'il offre de créer automatiquement, à partir de la représentation interne des documents, une forme acceptée par des systèmes de composition de texte. L'utilisateur de Mentor-Rapport n'a donc pas besoin d'apprendre à se servir de ces systèmes, et il est sûr que la mise en page de son document sera uniforme. Il lui suffit, pour composer son document, d'appeler la commande *compose* et d'envoyer le fichier produit par cette commande au système de

composition de texte disponible sur son site.

Actuellement, trois systèmes de composition de texte sont connus de Mentor-Rapport: le système *Compose* de Multics [Multics 79], le système *Nroff/Troff* de Unix[2] [Kernighan 78] et le système TEX [Knuth 79]. Dans les composantes textuelles du document, les changements de polices de caractères sont indiqués selon des conventions propres à Mentor-Rapport pour garantir l'indépendance vis-à-vis des systèmes de composition de texte. L'indépendance de la structure interne des documents vis-à-vis du système de composition utilisé est un élément important; en effet, grâce à cela, les documents peuvent être transférés d'un site à un autre sans précautions particulières.

Le style de la mise en page effectuée par la commande *compose* est déterminé par le modèle de document utilisé au moment de la création. Néanmoins, les utilisateurs qui connaissent le système de composition de texte utilisé, peuvent modifier la mise en page par insertion directe de commandes de composition. Le document est alors dépendant du système de composition correspondant.

Bien entendu, la commande *compose* tient compte des différents langages qui apparaissent dans le document. Les paragraphes qui sont des portes vers un langage structuré, sont formattés par le *décompilateur* de ce langage, de la même façon que lorqu'ils sont affichés à l'écran. La convention adoptée par défaut, pour la composition de ces paragraphes, est de mettre les mots clés en caractères gras, les commentaires en italique et tout le reste en caractères normaux. Cette convention est modifiable facilement par les utilisateurs.

2.4. Futurs développements: traitement des figures et des formules

Dans la version actuelle de Mentor-Rapport, la composition des formules mathématiques, des tables et des figures n'est pas effectuée. Cependant, tous les mécanismes de base nécessaires pour cela sont en place et aucun problème technique n'est responsable de cette lacune qui sera comblée prochainement. Pour comprendre de quelle façon la composition des formules, des tables et des figures, se conçoit dans Mentor-Rapport, il est utile de faire brièvement le point sur les systèmes de composition qui prennent en compte ces éléments. Par souci de simplicité nous classons ces sytèmes en deux catégories:

i) Les systèmes *non interactifs* tels que TEX [Knuth 79], TROFF (complété par ses préprocesseurs EQN [Kernighan 75] et TBL [Lesk 79b]), PIC [Kernighan 81] ou encore Ideal [Van Wyk 81]. Dans ces systèmes l'utilisateur fournit une représentation textuelle des formules ou des figures dans un langage ad-hoc. Lors de la composition du document, le système interprète cette représentation textuelle et la transforme en une suite de commandes destinées à piloter le périphérique d'impression. L'utilisateur doit sortir son document sur papier pour voir le résultat de la composition.

---

[2] Unix is a trademark of Bell Laboratories

ii) Les systèmes *interactifs* tels que EDIMATH [Quint 83] et PEN-MATH [Allen 81]. Ces systèmes utilisent un écran à haute résolution sur lequel les formules sont affichées sous une forme aussi proche que possible des notations mathématiques usuelles. Les formules sont saisies interactivement. Dans EDIMATH, les opérateurs mathématiques (fraction, racine, puissance, indice, intégrale, etc.) sont associés à des touches de fonctions du clavier. Les systèmes interactifs utilisent une représentation arborescente des formules. Ils fournissent des commandes interactives de manipulation de cette arborescence comparables à celles des éditeurs syntaxiques.

Bien sûr, nous ne prétendons pas faire une revue exhaustive des systèmes de composition de texte. Indiquons toutefois que, parmi les systèmes qui *ne traitent pas* les formules ou les figures, certains, comme Scribe [Reid 80] et Etude [Hammer 81] utilisent également des représentations structurées des documents, pour un traitement interactif dans le cas de Etude et pour un traitement non interactif dans le cas de Scribe.

Comme nous l'avons dit dans la section 1.4, nous n'envisageons pas, actuellement, de créer un système de composition de texte, formules ou figures, interne à Mentor-Rapport. La stratégie adoptée pour la prise en compte des formules et des figures est la même que celle adoptée pour le texte: la commande *compose* **compile** la représentation interne de ces objets en une forme acceptée par le système de composition visé. La commande *compose* est donc un *compilateur multi-cibles* dont les cibles sont précisément des systèmes de composition de texte, formules ou figures.

Le désavantage de cette stratégie est d'orienter Mentor-Rapport vers la composition de texte *non interactive*. Cependant, dans la future version sur écran à haute résolution, on pourra améliorer cet aspect en jouant sur les décompilateurs: ils interpréteront la structure interne et afficheront une approximation de la forme de ces formules ou figures.

Pour décrire les figures, nous utilisons actuellement le langage Flip [Kahn 81a]. Un interpréteur de ce langage, qui affiche les figures sur un écran couleur à haute résolution, a été réalisé. Nous prévoyons la construction d'un compilateur qui, à partir de la représentation interne du langage Flip, crée du code pour TEX ou pour TROFF (en utilisant le préprocesseur TBL).

Pour les formules mathématiques, la représentation interne que nous adoptons dans Mentor-Rapport est basée sur les mêmes principes que celle utilisée dans EDIMATH. Nous prévoyons ici aussi la construction d'un compilateur qui créera soit du code pour TEX, soit du code pour TROFF (en utilisant le préprocesseur EQN). Notons qu'une première version du compilateur qui crée du code pour TEX est déjà opérationnelle.

Cette stratégie ne lie pas définitivement Mentor-Rapport à des systèmes de composition ou à des langages de description particulier; en effet, les compilateurs envisagés sont écrits en utilisant l'interface mentionnée à la fin de la section 1.1 et sont donc externes au systèmes. Ils sont appelés par la commande *compose* lorsque c'est nécessaire. Nous visons plus particulièrement, à cours terme, les systèmes TEX et TROFF car ce sont les deux systèmes dont nous disposons actuellement.

Conclusion

L'édition de documents multi-langages nous paraît être une étape importante dans le domaine du génie logiciel car elle permet de représenter un logiciel par un document unique tout en respectant la spécificité des composants et la hiérarchie entre ces composants.

Mentor-Rapport, outre ses fonctionnalités propres (interface utilisateur adaptée au traitement des documents multi-langages, banalisation de l'utilisation conjointe de l'édition syntaxique et de l'édition de texte, liaison avec les systèmes de composition de texte), hérite de toute la puissance de Mentor dans le domaine de la manipulation symbolique de structures hiérarchiques [Kahn 81b]. Il est donc possible de construire des processeurs de calcul sur ces structures, en particulier pour effectuer des vérifications de cohérence entre les composants d'un logiciel, pour maintenir incrémentalement cette cohérence ou pour la rétablir lorsque c'est nécessaire.

La possibilité de composer les documents, pour leur donner une présentation agréable, ou même pour les publier, est aussi un point important. En effet, une présentation propre et uniforme est un facteur important pour la circulation des informations entre les personnes concernées et met souvent en évidence les irrégularités éventuelles.

La possibilité d'avoir plusieurs points de vues sur les documents permet d'en extraire certains composants (programmes, documentation, jeux de tests etc) pour leur faire subir des traitements spécifiques (compilation, impression, exécution etc). Il est, par exemple, possible de sortir des nouvelles versions de la documentation d'un logiciel sans effort particulier.

D'une manière générale, ces différents points induisent un mode de travail plus rigoureux de la part des équipes de développement, qui sont déchargées de certaines tâches ingrates grâce à des outils de traitement adéquats.

Parmi les développements actuels de Mentor-Rapport, certains sont liés au développement du système Mentor général, comme l'amélioration de l'interface utilisateur par l'utilisation d'écrans à haute résolution avec dispositif de pointage, ou la spécification de processeurs sémantiques pour les langages manipulés (vérificateurs de type, interpréteurs) [Despeyroux 84]; d'autres sont spécifiques à Mentor-Rapport comme le traitement des formules mathématiques et des figures, la généralisation des strutures des documents et le développement de bibliothèques de modèles de documents standard.

Mentor-Rapport est un système largement utilisé. Il est intégré à la version 5 de Mentor qui est opérationnelle sur plusieurs machines dont les principales sont: HB68/Multics, Vax/Unix, SM90/Unix, Data-General/MV10000 et Apollo.

Pour conclure, je tiens à remercier tous ceux qui ont participé à l'effort de développement de Mentor et de Mentor-Rapport. Je tiens à remercier plus particulièrement B. Lang qui a effectué la plus large part du développement des mécanismes multi-langages de Mentor. Je tiens à remercier également V. Migot et D. Vérove, dont l'aide m'a été précieuse pour la mise au point, la maintenance et la diffusion de ces systèmes, et D. Clément qui a réalisé un compilateur permettant d'utiliser le système TEX.

Bibliographie

[Allen 81] T. Allen, R. Nix, A. Perlis, *PEN: A Hierarchical Document Editor,* ACM-SIGPLAN-SIGOA Symposium on Text Manipulation, Portland, Oregon, June 1981, Sigplan Notices Vol. 16, No. 6, June 1891

[Despeyroux 84] T. Despeyroux, *Executable Specification of Static Semantics,* Semantics of Data Types, edited by G. Kahn, D.B. MacQueen and G. Plotkin, Lecture notes in Computer Science 173, Springer-Verlag (1984), pp. 215-233

[Donzeau-Gouge 83] V. Donzeau-Gouge, G. Kahn, B. Lang, B. Mélèse, *Outline of a tool for document manipulation,* IFIP, septembre 1983, Paris

[Donzeau-Gouge 84a] V. Donzeau-Gouge, B. Lang, B. Mélèse, *Practical Applications of a Syntax Directed Program Manipulation Environment,* 7th International Conference on Software Engineering, Orlando, Florida, March 1984

[Donzeau-Gouge 84b] V. Donzeau-Gouge, G. Kahn, B. Lang, B. Mélèse, *Documents Structure and Modularity in Mentor,* Proceedings of the ACM SIGSOFT/SIGPLAN Soft. Eng. Symp. on Practical Software Development Environments, Pittsburgh, Sigplan Notices, May 1984

[Estublier 83] J. Estublier, S. Krakowiak, J. Mossière, Y. Rouzaud, *Design Principles of the Adèle Programming Environment,* Proceedings of the International Computing Symposium on Application Systems Development (ACM), Nuremberg (March 1983)

[Feiler 81] P. H. Feiler, R. Medina-Mora, *An incremental programming environment,* IEEE Trans. on Soft. Eng. SE-7, No 5, Sept. 81, 472-481

[Gandalf 84] The Gandalf Project, *ALOE Users' and implementors' Guide (Fourth Edition),* Departement of Computer Science, Carnegie-Mellon University, Pittsburgh, November 1984

[Hammer 81] M. Hammer et al., *The Implementation of Etude, An Integrated and Interactive Document Production system,* ACM SIGPLAN SIGOA Symposium on Text Manipulation, Portland, Oregon, June 1981, Sigplan Notices Vol. 16, No. 6, June 1891

[Kahn 81a] G. Kahn, *Flip, Manuel d'utilisation,* Rapport Technique No. 2, INRIA, Juin 1981

[Kahn 81b] G. Kahn, *The Scope of Symbolic Computation,* Invited paper, International Conference on Fifth Generation Computer Systems, Tokyo, October 1981

[Kahn 83] G. Kahn, B. Lang, B. Mélèse, E. Morcos, *Metal: a formalim to specify formalisms,* Science of Computer Programming, North Holland, Vol. 3, No. 2, 151-188, Aug. 1983

[Kernighan 75] B. W. Kernighan, L. L. Cherry, *A System for Typesetting Mathematics,* Communications of the ACM, Vol. 18, No. 3, pp. 151-157 (1975)

[Kernighan 78] B. W. Kernighan, M. E. Lesk, J. F. Ossanna, *Unix Time-Sharing System: Document Preparation,* Bell System Technical Journal, July-August 1978, Vol. 57, No. 6, Part 2

[Kernighan 81] B. W. Kernighan, *PIC - A Language for Typesetting Graphics,* ACM SIGPLAN SIGOA Symposium on Text Manipulation, Portland, Oregon, June 1981, Sigplan Notices Vol. 16, No. 6, June 1891

[Knuth 79] D. E. Knuth, *TEX and METAFONT, New Directions in Typesetting,* Bedford, Massachusetts, Digital Press, 1979

[Knuth 82] D. E. Knuth, *The WEB System for structured Documentation,* Fourth preliminary draft, Version 0.97, Stanford University, July 1982

[Lesk 79a] M. E. Lesk, *Somme Applications of Inverted Indexes on the UNIX System,* UNIX Programmer's Manual, Vol. 2, Section 11, January 1979

[Lesk 79b] M. E. Lesk, *TBL- A Program to Format Tables,* UNIX Programmer's Manual, Vol. 2, Section 10, January 1979

[Mélèse 80] B. Mélèse, *Manipulation de programmes Pascal au niveau des concepts du langage,* Thèse de 3ieme cycle, Université Paris 11 Orsay, 1980

[Mélèse 81] B. Mélèse, *Mentor: l'environnement Pascal,* Rapport Technique No 5, I.N.R.I.A., Octobre 1981

[Mélèse 82] B. Mélèse, *Métal, un langage de spécification pour le système Mentor,* Technique et Science Informatiques (AFCET), Vol. 1 No 4, Juillet-Aout 1982

[Mélèse 83] B. Mélèse, *Mentor Rapport: Manipulation de textes structurés sous Mentor,* Rapport Technique No. 23, INRIA, Avril 1983

[Mélèse 85] B. Mélèse, V. Migot, D. Vérove, *The Mentor-V5 Documentation,* Technical Report No. 43, INRIA, January 85

[Migot 83] V. Migot, *Un Pascal Modulaire sous Mentor,* Thèse de 3ième cycle, université Paris 11, sept. 1983

[Migot 85] V. Migot, *A New User Interface for the Mentor System,* Rapport Interne, INRIA, Avril 1985

[Milner 84] R. Milner, *A proposal for standard ML,* Conference Record of the 1984 ACM Symposium on Lisp and Functional Programming, Austin, Texas, August 1984

[Mossière 82] J. Mossière, J. Raymond, Y. Rouzaud, *Représentation interne et manipulation des programmes dans Adèle,* Colloque Génie Logiciel AFCET, Paris, Janvier 1982

[Multics 79] *Wordpro New User's Guide,* In the documentation of the Multics System, CII Honeywell Bull, September 1979

[Quint 83] V. Quint, *An Interactive System for Mathematical Text Processing,* Technique et Science Informatiques (AFCET), Vol. 2 No 3, 1983, pp. 179-190

[Reiss 84] S. Reiss, *Graphical Program Development with PECAN Program Development*

Systems, Proceedings of the ACM SIGSOFT/SIGPLAN Soft. Eng. Symp. on Practical Software Development Environments, Pittsburgh, Sigplan Notices, May 1984

[Reid 80] B. Reid, *Scribe: a high-level approach to computer document formatting,* Seventh Annual ACM Symposium on Principles of Programming Languages, ACM/SIGPLAN-SIGACT, Las Vegas, January 1980, pp. 24-31

[Reid 83] B. Reid, *Scribe: Histoire et Evaluation,* Actes des journées sur la Manipulation de Documents, J. André ed., INRIA-Rennes, Mai 1983

[Reps 82] T. Reps, *Generating Langage Based Environments,* Technical Report 82-514,Cornell University, Ithaca, NY, August 82

[Schroeder 84] A. Schroeder, *Integrated Program Measurement and Documentation Tools,* 7th International Conference on Software Engineering, Orlando, Florida, March 1984

[Sigplan 81] Procedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation, Portland, Oregon, June 1981, Sigplan Notices Vol. 16, No. 6, June 1891

[Sigplan 84] Proceedings of the ACM SIGSOFT/SIGPLAN Soft. Eng. Symp. on Practical Software Development Environments, Pittsburgh, Sigplan Notices, May 1984

[Stallman 81] R. M. Stallman, *EMACS The Extensible, Customizable Self-Documenting Display Editor,* ACM SIGPLAN SIGOA Symposium on Text Manipulation, Portland, Oregon, June 1981, Sigplan Notices Vol. 16, No. 6, June 1891

[Stromfors 81] O. Stromfors, L. Jonesjo, *The Implementation and Experiences of a Structure-Oriented Text Editor,* ACM SIGPLAN SIGOA Symposium on Text Manipulation, Portland, Oregon, June 1981, Sigplan Notices Vol. 16, No. 6, June 1891

[Teiltelbaum 81] T. Teiltelbaum, T. Reps, *The Cornell Program Synthesizer: A syntax directed programming environment,* Communication of the ACM, vol. 24, no. 9, September 81, 563-573

[Van Wyk 81] C. J. Van Wyk, *A Graphics Typesetting Language,* ACM SIGPLAN SIGOA Symposium on Text Manipulation, Portland, Oregon, June 1981, Sigplan Notices Vol. 16, No. 6, June 1891

[Verges-Escuin 73] J.C. Verges-Escuin, J.P. Verjus, *Reconnaissance automatique des structures des textes en vue de l'édition,* R.A.I.R.O. Octobre 1973, pp. 85-120

[Walker 81] J. H. Walker, *The Document Editor: A Support Environment for Preparing Technical Documents,* ACM SIGPLAN SIGOA Symposium on Text Manipulation, Portland, Oregon, June 1981, Sigplan Notices Vol. 16, No. 6, June 1891

[Wood 81] S. R. Wood, *Z - The 95% Program Editor,* ACM SIGPLAN SIGOA Symposium on Text Manipulation, Portland, Oregon, June 1981, Sigplan Notices Vol. 16, No. 6, June 1891

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique